A comprehensive guide:

the void

Table of Contents

-RETIRE, 24th of August 2018;

Please do provide the appropriate credit when sharing this article.
Here are the links to my Youtube Channel and my Discord Server.

1. The void: basics & misconceptions
    - What is the void?
    - How does save/ resetting affect the void
2. Position in RAM
    - Vertical and Horizontal movement.
    - The importance of outdoor voids
    - Calculating steps between 2 addresses
3. BSOD
    - Why do BSOD's occur and how do you avoid them
4. Routing & walls
    - Walls
    - Mapdata
    - 3D model data
    - Itemdata
5. Routing to indoor Maps
    -Getting to the Matrixdata of the Map
    -Getting inside the Model of the Map
6. Battletower voids
    - What makes them so unique
    - Routing in Battletower voids
7. Fake Sinnoh
    - What are they
    - Why do they exist
    - (Almost) obsoleted
8. Black Sinnoh
    - What are they
    - Why do they exist
    - What are their uses

9. Alt-RETIRE routing
    - What is Alt-RETIRE
    - Writing tiles into the void
10. Wrong Warping
    - How Wrong Warping changed voiding as we know it
    - Coordinate warp
    - Wrong Warp
    - Routing using Wrong Warps
    - Map 35 CollisionCorruption routing

1. The void: basics & misconceptions

1.1 What is the void?

The best way to describe the void from a technical perspective would be;
"All bytes in RAM, represented as a walkable space."

The void is just us walking through RAM, in a completely predictable manor.

1.2 How does save/resetting affect the void

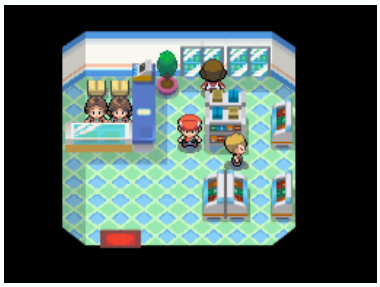
First we have to talk about the MatrixLayout, which is the area accessible within regular gameplay.
We can change our MatrixLayout and therefore its width by warping to/ save resetting in a Map.
Each Matrixlayout has its own Matrix_width.

There are 558 Maps, but there is a smaller amount of MatrixLayouts since Outdoor Maps are grouped together.
Let's split the types of MatrixLayouts into three groups.

1.2.1 Sinnoh / Outdoor Maps with Matrix_width 30

All outdoor Maps are a part of Sinnoh, which has its own giant MatrixLayout.
Therefore it doesn't matter in what Outdoor Map you save in, they will all bring you to the same Outdoor void. They will all have a Matrix_width of 30, which is important later on.

1.2.2 Indoor Maps with Matrix_width 1



The majority of Maps that aren't a part of Sinnoh / Outdoor Maps have a Matrix_width of 1.
They also all have their unique MatrixLayouts, since none of them are connected.

1.2.3 Indoor Maps with Matrix_width >1

Then we have our exceptions. These are Indoor Maps that, because they're bigger than a regular Indoor Map, they require a bigger Matrix_width.
Think about Caves, Amity Square, Pal Park and so on.
Obviously, they also have their unique MatrixLayout.

2. Position in RAM

We have established that the void is RAM. The game reads your Map ID based on a 2byte value in RAM. The bytes it reads from is dependent on your coordinate offset in correlation to MatrixLayout. To put it simply, by moving in the void we manipulate our location in RAM, starting at our MatrixLayout. If we move to a 2byte value which is '0008' our Map ID will become 8.

2.1 Vertical and Horizontal movement.

In order to route you must understand exactly how moving in specific directions manipulates your position in RAM. Here we have the recalculation used to calculate how many bytes we move by taking an amount of steps.

```
x_offset = floor(x / 32) * 2
y_offset = floor(y / 32) * (2 * matrix_width)

current_offset = MatrixLayout_start + x_offset + y_offset
```

This calculation was provided by Ganix.

Since a Map is 32x32 tiles, we divide our coordinate by 32, now we have the amount of Maps we move.
But since one Map in RAM is a 2byte value, we multiply by 2.

Therefore, moving 32 steps along the x_axis will move you 2 bytes back or forward in RAM.

There's one difference for the Y offset. Instead of just doing *2 because Maps are a 2byte value, it does 2*matrix_width. As we said before, there are three kinds of Maps. Outdoor Maps always have a Matrix_width of 30, while Indoor Maps either have 1, or by exception a value between 2-4.

Therefore, moving 32 steps along the y_axis will move you 60 bytes In Outdoor Maps, 2 bytes for Indoor Maps with a matrix_width of 1 and 4-8 bytes for the exceptions.

2.2 The importance of outdoor voids

Because moving vertically moves you 60 bytes/Map in Outdoor voids you should almost always move along the y_axis when traversing to a specific location in RAM. Just think about it like this. Reaching Mapdata only takes around 2300 steps downwards for Outdoor voids.

If you would try to go right, you'd only move 2 bytes/Map. This means it would take you 30x as long. That would mean it'd take you 69 000 steps instead.

This would also apply to going down in indoor Maps with Mapwidth 1, and the length would become shorter when using Indoor Maps with slightly higher Mapwidths.

2.3 Calculating steps between 2 addresses

Since we know how steps convert to bytes, we can obviously reverse our calculation to let us find the number of steps between 2 addresses in RAM. Since we always start at MatrixLayout_start we will use this as our starting location in my example. And as our destination we'll use Mapdata.
We know it should be around 2300 steps.

MatrixLayout is located at `Base + 0x22ADA = Base + 142042`
Mapdata is located at `Base + 0x23C6E = Base + 146542`

Here we have a formula I wrote that allows you to find the steps in between any 2 addresses.

Along x_axis (horizontal movement)

```
(DestinationAdrr - StartAdrr) / 2 * 32
```

Along y_axis (vertical movement)

```
(DestinationAdrr - StartAdrr) / (2 * Matrix_width) * 32
```

Let's fill it in according to our goal.

```
(Mapdata - MatrixLayout) / (2 * 30) * 32
```

And now we'll fill it in with the correct values.

```
(146542 - 142042) / (2 * 30) * 32 = 2400
```

Because this value is positive, we have to go 2400 steps South; if it were to be negative you'd have to go North.

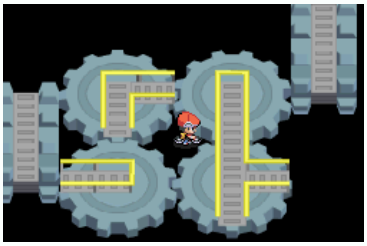
3. BSOD

A Black Screen Of Death or BSOD is a common occurrence in older voiding routes.
Last year I found a Map with very interesting properties, Map 35. This is a Map that due to loading 3D Models caused collision to mess up, giving you what essentially was Walk Through Walls. When further investigating this intriguing Map the cause of BSOD's was found. The same 3D Models causing Map 35's Walk Through Walls caused crashes when refreshing graphics. You can remove the Models by losing a battle, using elevatorwarps and similar warps or entering the Hall Of Fame. *Read 10.4 Map 35 CollisionCorruption Routing for more information.*

Map 35 isn't the only Map that loads 3D Models. We're referring to specific Models that once loaded never unload. I added all Maps that cause BSOD's in the void.lua, usually they occur because of elevators, but some Maps like Wake's Gym and Volkner's Gym load very specific models unique to their Maps. This is what the models would look like if you could refresh graphics and get to (0;0) where the Models are located.

Elevator

Wake's Gym (122)



Volkner's Gym (154-155-156)

Byron's Gym (35)

4. Routing & walls

Now we can get into routing. While it is possible to use any value in RAM, getting to them can sometimes take a very long time or be based on portions of RAM that might differ between games because of different save data, flags having been set and so on. Luckily, we do have some areas in RAM that we can get to consistently across games, regardless of progression, and are completely manipulatable.

Before we go into specific routing I must tell you about Map Id's above 0x22E, or 558 in decimal.
Since we read Maps directly from RAM they can go far above the hardcoded maximum.
To cope with the potential dangers of Maps reading from unintentional areas of RAM, the developers put in an error handler. Here's the code for it.

```
uint32 ValidateMapID(uint32 map_id)
{
  if ( map_id >= 0x22F )
  {
    ErrorHandler();
    map_id = 3;
  }
  return map_id;
}
```

This changes all Map ID's equal to or higher than 0x22F into Map ID 3, Jubilife City.
Therefore you can use all Maps with an ID of 559 and over to change your void to Outdoor by save resetting.
Since the Map is now Jubilife City, it will also hold the scripts, functions, furniture, objects, warps and triggers of Jubilife City.

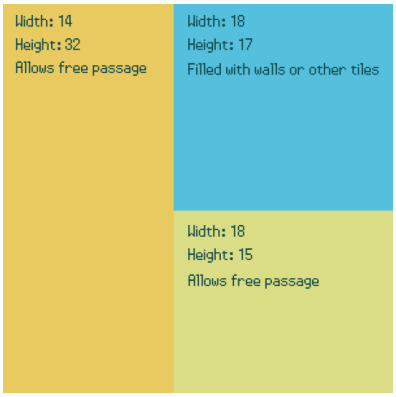
4.1 Walls and other tiles

The void is filled with walls and tiles, which are also read from RAM. It's hard to keep track of your position in RAM for walls to dodge them, but in most cases they aren't a huge deal as long as you know the general layout.

When going North starting at MatrixLayout walls aren't a problem, but since that's barely used anymore except for Itemdata or values easily accessible around the Matrixlayout, that's not too useful.

Usually the wall/tile layout loops every 32 steps, or one Map.
They do get slightly influenced by crossing loadlines, but this shifts the tiles rather than changing the entire layout.

When going West and South to reach Mapdata the Wall/tile layout becomes harder to work with, since it takes up a big percentage of a 32x32 Map.
Therefore, I made a visual representation of the Tile Layout inside one Map.



As you can see the left and bottom side of Maps are completely walkable.
Usually you move in multiples of 32 steps since that will always put you at the same point in a Map's layout. When routing with Alt-RETIRE I will go more in-depth with the act of writing specific tiles in the Wall zone marked blue.

4.2 Mapdata

Mapdata is the most frequently used portion of RAM for routing. It's only 2400 steps from Matrixdata, allows you to get to ANY Map and honestly isn't too hard to work with. It still has its flaws, which is why I will also go into the other two portions of RAM we utilize.

So, what is Mapdata exactly? This is the portions of RAM that is utilized for currently loaded Map objects, furniture, warps and triggers. This means this portion of RAM is overwritten the moment you enter a new Map, which would load the Mapdata of the Map you entered.

By chaining the values written in Maps you will eventually end up at the Map you want.

The reason this is possible is because the Mapdata RAM doesn't get wiped when leaving a Map, only the portions that get overwritten by the new Map will change. Therefore, if you enter Map 0002 (underground) which overwrites most all Mapdata, then enter Map 0000 (Mystery Zone "Everywhere") which holds nearly no Mapdata, you will only overwrite the top values in Mapdata.

4.2.1 Mapdata Routing; Requirements

In order to route with Mapdata you will need 4 files.

The first was provided by MKdasher, it is a Lua script called void.lua.
It's a RAM-viewer with additional features to help with voiding. First, you can label any range of Maps according to their type, and they will be highlighted on the RAM-viewer. The RAM-viewer also centralizes onto the player's current location in RAM, which is highlighted white. You can also put it on 'manual' mode to check Map ID's that aren't on screen, similar to a normal RAM-viewer.

The second file is Map_Events.txt, provided by Ganix.
This is a dump of all the Mapdata. You can search for specific values in this dump, and it will show what Maps write these values, be it using furniture, objects, warps or triggers.

The third is Map_scripts, also provided by Ganix. This lists all Maps in the game, including all the scripts in runtime and their corresponding script number. This can be used to find your Map ID's while simultaneously being able to find scripts accessible using the RETIRE menu or Alt-RETIRE glitch.

Fourthly I recommend the Tile Layout image as guide, since it applies to the Mapdata section.

4.2.2 Traversing to Mapdata

First we have to get to Mapdata itself, here's the route you utilize to get to the top of Mapdata.
From Poketch Co.

1 S
17 W
14 N
574 W
save reset
2097 S
19 W
288 S

The red steps traverse you through RAM until you hit a Map with ID above 558, therefore becoming Jubilife. It's also located perfectly in line to avoid Maps that cause movement or BSOD's when you have entered them.
Read 3. BSOD for further information on the subject.

4.2.3 Mapdata: The routing

If you haven't already you should load up the void.lua provided by MKdasher.
I recommend pressing the arrow in the top right corner to change the RAM-viewer to the top screen.
I also recommend pressing 'menu', and selecting 'Hex' to turn the values into their decimal equivalent.
This way you can see the amount of steps you take. Since it is a Lua script, you must use an emulator to utilize it. I Personally recommend DeSmuME since it's easy to use, despite it not being as advanced as No$GBA.
It's perfect for routing, and I think it gets too much hate for the accessibility it provides.

Your screen should currently display the overlay as shown.
The white highlight is the Map you currently reside in.

At the bottom screen you can see what the colors of Maps correspond to. You can completely customize the Names, Colors and Maps within the document when needed.

I should have added all BSOD-causing Maps, or ones that might softlock you. There might be a couple I forgot to add.

Now click the 'edit' button on Desmume's Lua window to edit the file, I will be showing the process utilizing Notepad++. You can use any preferred text editor.

First you open up the Map_Scripts file and search for the Map you'd like to route towards and put the Map ID between the { } in the Highlight section.

After that you open the Map_Events file and press CTRL+F and search for the Map ID.

I recommend putting a SPACE before and after the ID in the search function.
This way you avoid finding 65235 when searching 23.

If you find your value under 'INDICES' they won't be usable, so try again until you find it elsewhere. Usually you have many choices, put their Map ID's between the { } of Highlight2.

```
-----------------------------------------------
local mapId = {
    Highlight = {
        color = '#f7bbf3',
        number = {}
    },
    Highlight2= {
        color = '#F80A55',
        number = {}
    },
    MysteryZone = {
        color = '#88888866',
        number = {0}
    },
    Blackout = {
        color = 'orange',
        number = {332, 333}
    },
    Movement = {
        color = 'purple',
        number = {117, 177, 179, 181, 183, 192, 393,
            474, 475, 476, 477, 478, 479, 480, 481, 482, 483,
            484, 485, 486, 487, 488, 489, 490, 496}
    },
    VoidExit = {
        color = 'yellow',
        number = {105, 114, 337, 461, 516,186,187}
    },
    BSOD_DANGER = {
        color = 'red',
        number = {35, 88, 91, 93, 95, 115, 122, 150,154,155,
        156, 176, 178, 180, 182, 184, 185, 188}
    },
    Extra = {
        color = 'red',
        number = {}
    },
    Jubilife = {
        color = '#66ffbbff',
        number = {3}
    },
    Normal = {
        color = '#00bb00ff',
        number = {}
    }
}
------------------------
```

I will be showing off a route to get to Map 251; Pal Park R1-01 (Outdoors).
My name is RETIRE after all.

There are three Maps that hold the value 251, but only one that doesn't have it under 'INDICES', which you cannot use. This Map Is 393; Pal Park (Indoor).
This makes sense, since it has a warp directly to the Map.

```
WARPS
    Warp 0:
      X Coord      [   7 ]
      Y Coord      [  19 ]
      Map ID       [ 392 ]
      Type         [   1 ]
      -            [   0 ]
      -            [   0 ]
    Warp 1:
      X Coord      [   7 ]
      Y Coord      [   7 ]
      Map ID       [ 251 ]
      Type         [   0 ]
      -            [   0 ]
      -            [   0 ]
```

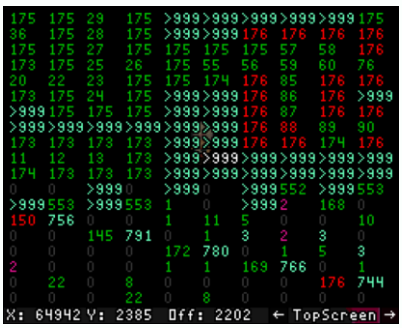
Here is the dump of the Warpdata of Map 393.

But we do have to reach Map 393 before that, so we need a Map that will write a 393.
Not coming to a surprise, you can use Map 392, Route 221.

This is the route with a warp to Map 393, and obviously Map 393 also has a warp leading back to Map ID 392.

392 will be written by Map 0002, Underground. Lower values like 2 are written by almost all Maps, including our starting Map Jubilife City. So now we should be able to get to Pal Park Outdoors's Map by reversing the order of Maps we enter.

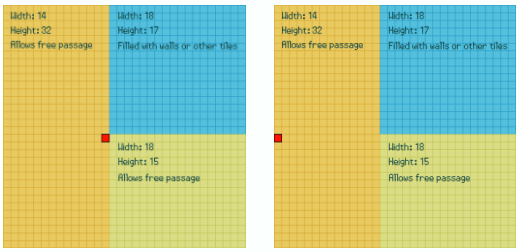
Now I put 251 into Highlight and 393, 392 and 2 in the 'Highlight2' section.
Now these values are Highlighted so I can easily access them.



Now that I have put in the values, I can see that Map 2 can be accessed to the right and left of me. I do already know that Map 392 will be written in the same column as I'm currently in, so going the least amount of columns away is the most efficient.

So I must go 2 Maps right. One Map is 32 tiles wide, so going 64 steps right will put me in Map 2. But, I'm not entirely at the left of my current Map, since, to avoid walls, I only had to go to the middle of this Map.

Let's take a look at the Wall_Layout, I marked my current location with a red square. If we're currently here, going right until we reach the end of our current Map would take 18 steps, then 1 extra to reach the next Map. After that we do another 32 to reach the next Map.
18+1+32 = 51 steps E.



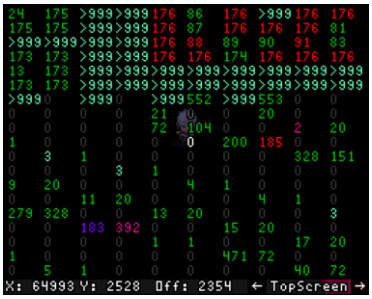
Now I've moved myself 4 bytes further into RAM, and have positioned myself straight above the Map 2's. The first Map 2 is 3 Maps further, the second 5 Maps. 180 bytes and 300 bytes further in RAM respectively.

One small issue is that we'll pass Map 552 if we want to get to Map 2, which will load new Mapdata into RAM. Luckily for us, Map 552 holds a negligible amount of Mapdata.

Therefore it will only overwrite a small portions of RAM, enough to overwrite the Underground 180 bytes away, but not enough for the Underground 300 bytes away.

If we want to get to Map 2 we'll have to calculate the amount of steps to take.
First we take 14 Steps south to reach the end of our Map. We take 1 Step extra to reach the next Map. Now we still have to go 4 Maps, so that's another 128 steps.

I went 143 steps (14+1+ (4*32)) South, and reached Map 2 (Underground).
At the right we see our position in the Tile Layout and in RAM.
This wrote our Map 392 into RAM, but it also loaded way more data than just that.



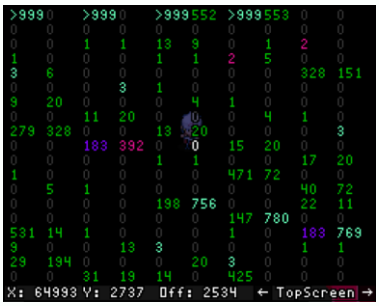
What's interesting to note is that we are now inside a Mystery zone,
or at least the byte we are currently located at has 0 as value.
We overwrote the Map we are currently standing in.
The game won't update our current Map until we move though,
so we're essentially still in the Underground Map,
but the entire Map around us is the 00.

Back on topic, the Map 392 is located further in RAM.
356 bytes to be exact. The easiest way to get there is to go 360 bytes (6*60)
further in RAM, and go 4 back.

That means I have to go 6 Maps further in RAM, and an additional 17 to position myself past the walls, to traverse left afterwards.
This equals to 209 steps (17+(6*32)) South.
Our last steps were also going south, 143 steps.

Since it would be odd to write '143 S – 209 S' we group them up together.
That totals up to 352 (143+209) steps.

This is our position in Walls and in RAM once we get there.
Since we're at the left in the Tile Layout and we put ourselves at the perfect height to dodge the walls, all we have to do is 1 step West to reach the next Map, and another 32 to reach the Map in which we wrote 392.



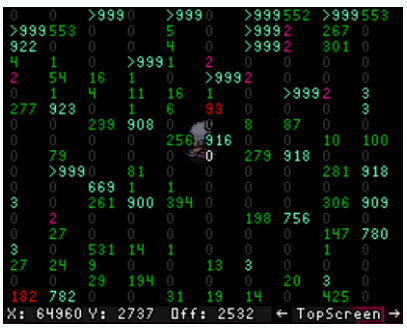
That totals up to 33 steps West. Just like earlier we'll be overwriting our own Map when entering it. This time you're clearly still inside the Map, since you can open your Menu's. In this case that's even useful!

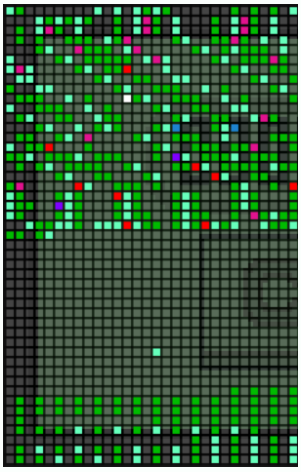
We don't use it often, but if you truly require it you could save reset here since Map 392 is an Outdoor Map, meaning our position in RAM stays the same.
When resetting the game will wipe all leftover Mapdata, only loading the Mapdata required for Map 392 itself. This could make traversing slightly easier, but usually it is faster just trying to dodge additional Maps.

After we go 33 W this is our current position in the Tile Layout and RAM.
We entered Map 392 and clearly overwrote the Mapdata, but there's no Map 393 in sight. This means that it is probably off-screen.

To see values that are located off-screen you press the 'CENTER' button and toggle it to 'MANUAL', the 4 directional arrows should light up yellow. By pressing one of the arrows, the RAM-viewer will shift accordingly. By going right and left you will eventually stop moving, that doesn't mean you can't go further.

If you're at the end of the RAM-viewer's capabilities at the left side, going a Map left would still mean going 2 Maps backwards in RAM. The Map you'll enter is the one located at the right of RAM-viewer and one row up. When going right it is one row down.

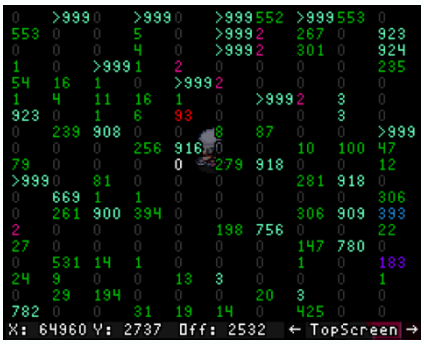




In order to know whether the value is located at the left or right, you can also zoom out the RAM-viewer.
You press the Arrow next to 'TopCcreen' and it will show the values, but only sorted by color, you won't see the values. You should probably change the color of the value you are currently looking for to avoid seeing the wrong values.

In this case the value I'm looking for, 393, I highlighted it blue.
I can see two blue Maps to the right to me, makes sense, there are two warps from Route 221 to Map 393.

Now I go back to the 'TopScreen' Layout and move shift the RAM-viewer to the right.
In this case the first Map 393 is only 2 bytes offscreen, or one Map to the right.

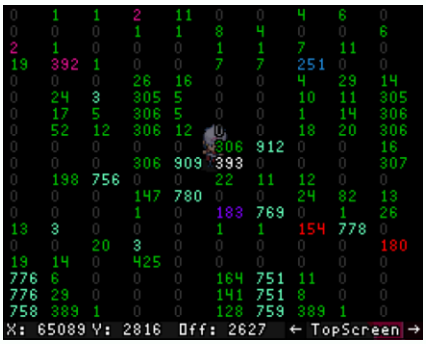


The closest Map 393 is 190 bytes further in RAM.
The easiest way to get there would be going 3 Maps South (3*60=180) and an additional 5 Maps East (5*2=10).

Here we stumble upon a small issue. We would have to pass Map 306 and 909.
Map 306 doesn't hold enough data to overwrite 393, but Map 909, converted to 3 by the ErrorHandler, certainly does. We can also enter the Map from above, then the steps would be: 2 Maps South (2*60=120), 5 Maps East (5*2=10), 1 Map South (1*60).

We can't go directly down since we're located above Walls, but going one right puts us in the next Map, 297. This Map won't overwrite the Map 393 and we can go down after doing that.
So in the end, we're going 1 Map East (1*2), 2 Maps South (2*60), 4 Maps East (4*2), 1 Map South (1*60).

Since I'm hoping you are starting to understand the Wall layout, I will just show my current position in RAM for the remaining steps.



When converting everything over, these are the steps to follow from this point.
1 E, 78 S, 128 E, 1 S.
I did choose this particular route for a specific reason.
There's a Mapscript that runs the moment you enter Map 393.

Once it runs, it checks if a flag has been set, if not it runs a cutscene with Proffesor Oak. He gives you the Poketch App, Moves you one tile right, then leaves and the flag is set so the cutscene won't occur again. 2 Problems here, the first is that the cutscene moves you, the second is that it won't if you've already done the cutscene.

We have to make sure to implement that additional step in the route,
but make a division between the script running and the script not running.

Also, since we have a Mapscript running it's better to write '1 S 17 S' instead of '18 s'.
The game won't recognize the step if a Mapscript runs, so going 1 S would show 00.
Above us the blue Map 251, our end goal, is written. So I'll go a bit more south to dodge walls, go 2 Maps right and go up until I hit Map 251. The Maps under Map 251 do not hold enough data to overwrite it.

These are the steps up until Map 251; starting at the last section.
17 S
63 E if Oak cutscene ran. 64 E if the cutscene didn't run.
177 N
1 N

Now that we can route to any Map ID, how do I get to the Matrixdata or physical Map?
Read: *5.2 routing to indoor Maps* or *6.2 routing using Battletower*.

4.3 3D model data (Advanced)

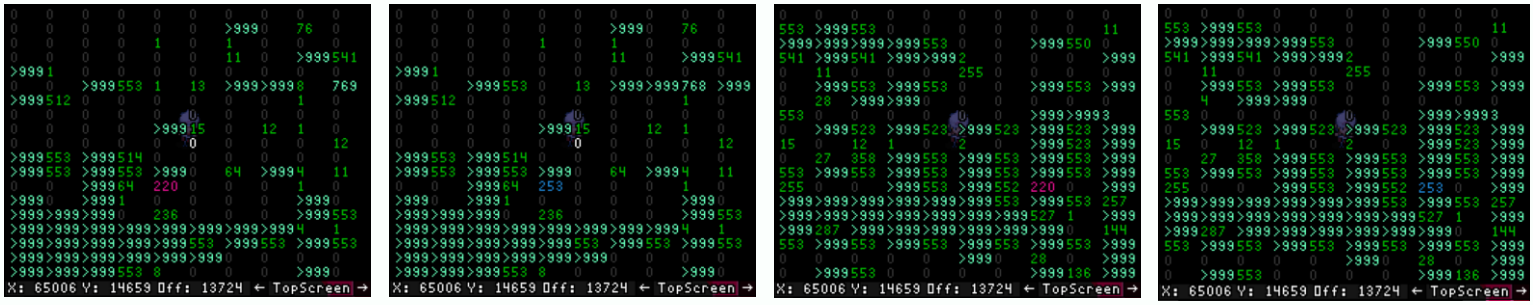
Since we got to our goal using Mapdata we'll go ahead and take a look at 3D model data.
3 D model is similar to Mapdata, except the data is way more volatile to work with, but you can write values very far into this RAM section by refreshing graphics in a Map. The data will be rearranged and some values will be written when doing so. Ganix dumped 3D model data, but it is nowhere as easy to use as Mapdata. It is very similar in the workings, except you have to refresh in the Map instead of just entering. I don't actually use the document myself anymore since I found a second, questionably easier, method to write to 3D model data.

What I like to use is a copy function. When opening a textbox of any kind the game copies your current Map ID into 3D model data. It is very high up so you'll have to avoid a lot of Maps. Once you enter it, the data will most likely be overwritten. Since we are just in a 1x1 version of the Map, we can now open a textbox again to write the Map ID into the byte again. And since by moving we won't be loading more data, we now have a 32x32 Map.

Opening the pop-up menu will suffice, in Maps you can't open your Menu you can use a Key item that pops up text as well. To make this work I first route to a Map in Mapdata, then travel to 3D model data and the byte will be copied there.

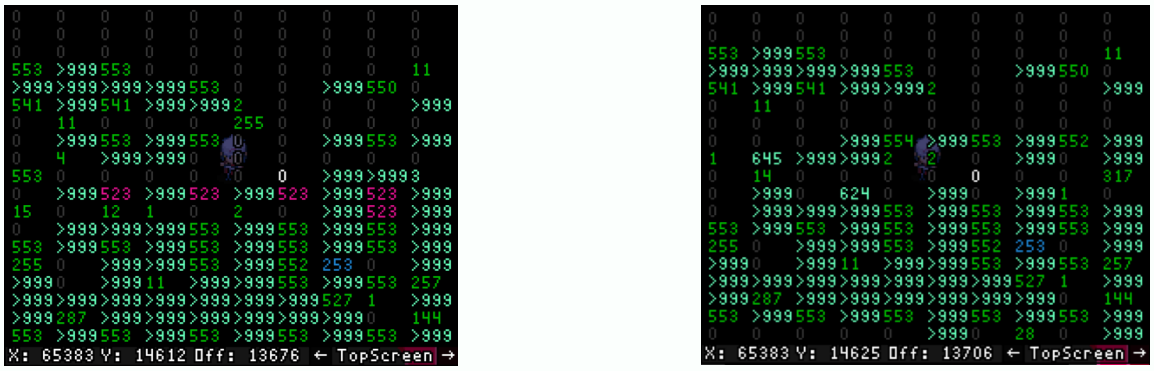
Honestly writing a completely guide on 3D model data would be too much work. I will add in the dump of 3D model data by Ganix but it requires you to match the Mapnames within the code to the actual Mapnames and their ID's.

Here you see a 'before' and 'after'; First I open a menu in Spear Pillar (220) and then in Amity Square (253)



The 3D Model section is located way further in RAM than Mapdata, about 11 500 steps. The locations at which the copy function copies your current Map to are located around Y: 14 760. If it is hard to traverse to the Maps without overwriting the data I recommend searching a Map 523, which overwrites the top of this RAM section slightly so you may enter the copied Map at the most right. Map 523 gets written above this section consistently when a textbox displays, but if you've overwritten it you will probably still find one somewhere at the top of this RAM-section.

Now that the top section is opened up you can usually go through the Map 553's and Map 552's to enter the Map you copied. Once you enter it the RAM-section will most Likely be overwritten so don't forget to re-write the Map by opening the side Menu.



4.4 Itemdata

Here we get to the most user-friendly way of routing. Both Item ID's and Item amounts are both 2 byte values, they're perfect for routing. You can completely modify the locations on the fly by tossing Items and write multiple Maps next to each other in a grid. Since Mapdata and 3D model data exists there's almost no use for it for routing, but there is one particular route I had to write using this. A route which allows the user to catch Arceus infinitely, in any Map in the game and with a completely modifiable locationtag. The issues with this routing method is the cost of items, and the amount of steps you have to take to get to Itemdata. Just to get to Itemdata you have to take 75 000 steps going North.

This means you have a chance of crossing BSOD Maps or elevators, and we can't even Wrong Warp to this Location. First, let's calculate the exact difference in between the Address of the MatrixLayout and the Itemdata, and convert it to steps like I showed earlier.

Matrixdata is located at `Base + 0x22ADA = Base + 142042`
Itemdata is located at `Base + 0x00838 = Vase + 2104`

Calculation: `(2104 - 142042) / (2*30) * 32 = -74 624 steps`

Since our outcome is negative, we have to go North.
I wrote a route that puts you in the middle of your Medicine slot Itemdata, which is located 54 bytes after the items in the Mail slot. This is what you would use, since Grass and Heart Mail only cost 50₽ and there are some medicine that only cost 100₽. As I said I've only ever used this once, for the Arceus route. I'll just show the route and from what you've read before you should be able to figure it out from there. The only additional knowledge you might need is that the ID is stored first, then the amount of the Item. This Bulbapedia article lists all items and their ID in Hexadecimal and decimal values.

Just be realistic with the items you're using and their amounts/costs. You don't want to end up with a 1.000.000₽ total cost, like I did when I wrote my first version of the Arceus route. Got it down to 80 860₽ though.
Also, if you were planning on trying to route using Pokémondata, this is impossible.
First off it's located above Itemdata, even if it's only by a 1000 steps. Second and most importantly, the data is encrypted. The values are almost all equal to or above 559, resulting in a Jubilife City. Itemdata is just far superior.

---
1 S
5 E
450 N
Save reset
64 W

73500 N
---

5. Routing to indoor Maps

5.1 Getting to Matrixdata of the Map

After you have routed towards the desired indoor Map ID you probably want to get to its Matrixcenter. Getting to the Matrixcenter is usually not a big deal, since you just have to save reset and then travel towards 0;0. There are quite some indoor Maps that crash when you move East, with currently unknown cause.
There's no way around this other than trying to get the Map ID of that Map to the right, above the Matrixcenter instead. That way you only have to move South and West, obviously this means you cannot use Mapdata to reach these particular Maps.

These exception aside reaching the Matrixcenter isn't the big issue. Obviously there are walls surrounding the walkable area inside the Map, preventing you to move inside. There's an exploit that can get you through walls, but it only works in some cases, when the walls aren't too thick. By refreshing graphics at at underflowed coordinates, you can cause the collision for walls (and other tiles) to change slightly, allowing you to move through them in some cases. If you hit a wall inside the Map itself with positive coordinates you should either try again with graphic reloads at different positions or give up as it's unlikely you will be able to enter the Map.

Let's take Map 7 as an example. We start off strong, it's one of the Maps that causes crashes by moving right in their void. Luckily 7 is a value that is easily accessible in the RAM that can be located to the right of the Matrixcenter.

5.2 Getting inside the model of the Map

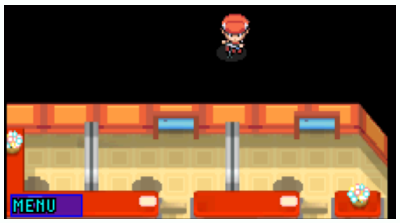
Steps to get there are
1 S
5 E
430 N
save reset (Jubilife void)
64 N
203 E

save reset (Map 7, located North and East of Matrixcenter)

214 W (X value = 10)
479 S (Y value = 65535)
You can now see 3 NPC shadows loaded

Graphic reload, because of our position this will slightly alter the wall layout allowing us to go down.

After the graphic reload the NPC's are currently not visible, but if we tilt the camera slightly, we'll notice that their shadows just happen to be hidden behind the counters.
You can also see them without adjusting the camera but they aren't easy to notice.

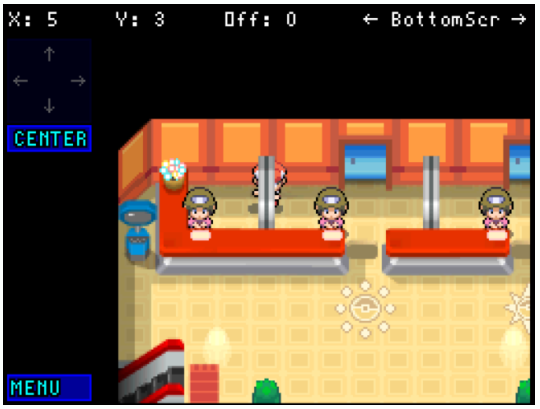


Now you go south, you'll go partially through the wall and hit a different wall. Not quite far enough to enter the Map, but just 1 tile above what usually would have no collision.

If you go too slow you'll notice that you stop at Y = 2, you have to go fast enough to trick the collisiondetection.

To change it to the collision it usually would have, refresh again. Because our Y-value is now 1 and therefore within the normal Mapmodel, the collision will fix itself.

Now you can go 1 down into the wall, because the tiles above the iron bars don't have collision. The bars themselves also do not. You can also use the doors since they also do not have collision, but the middle one in this case cannot be used. Not because it won't let you step into it, but because due to how the union-room is coded warping without conversing with the NPC results in a crash.



Depending on your Map you might have to enter from above, from the left side or it could also be impossible to enter at all.

6. Battletower voids

We've spoken about a multitude of Map types, but there always seem to be something extra that a Map can have to make things a little more difficult, useful or unique. In this case all of the above apply. Battletower voids are the voids you enter when saving in any Map that is part of the Battletower Map ID list. These include; 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 493.

There are 3 known ways to actually save at coordinates of outdoor voids anyway, with the Map Id of an outdoor void. This is the first listed, the other two include using
7. Fake sinnoh's or 10.4 Map 35 CollisionCorruption routing.

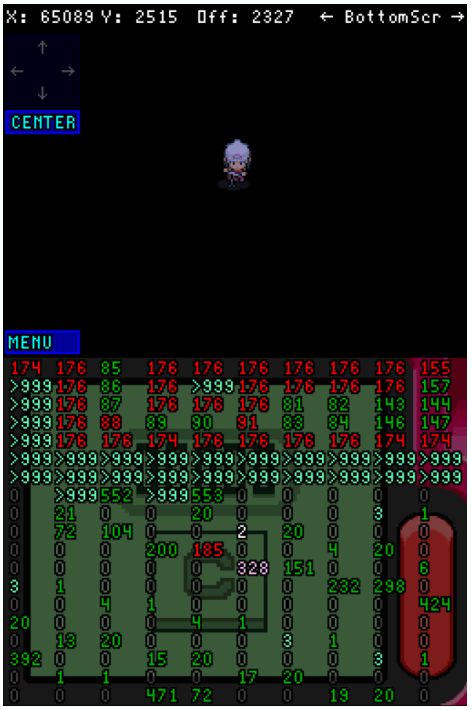
6.1 What makes them so unique

Whenever you are in a Battletower void you can forgot some of the essential parts of voiding because these voids have some special properties. First of all, bytes shift by 8000. This itself is useful for some glitches like ASLR scriptcalling, but besides that it also changes some fundamental parts of the void. Whenever you're usually in a void there are a couple of things preventing you from saving directly at the coordinates of outdoor Maps.

The first is a giant barrier starting at (0;0) which extends both vertically and horizontally. Next to that, the RAM-section after Matrixdata of Indoor Maps are all 00's, and in Outdoor voids its obviously the Matrix itself, or Sinnoh. Despite 00 being considered an outdoor Map, you can't open any menu here so it's quite useless.

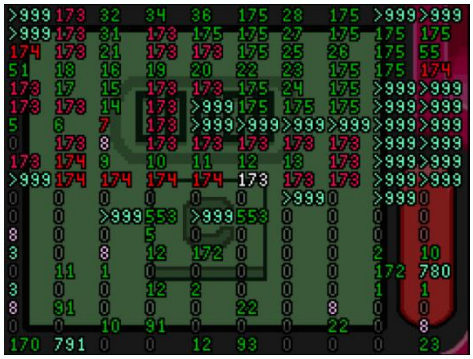
And that's why I said that inside battletowers, you have to forget the fundamental rules.
Inside these voids, the barrier is no longer blocking you from passing and the game allows you to open your menu inside of Mystery zones. This means that if you save in those, you'll end up in Sinnoh.

6.2 Routing in Battletower voids



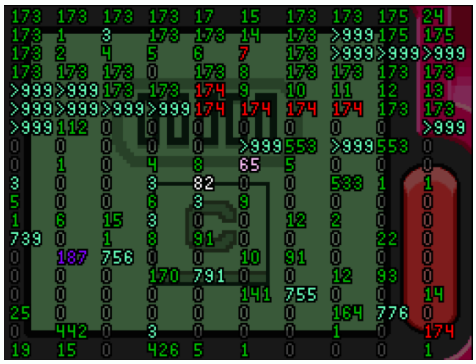
There's always a price you have to pay for something good though. In this case, that is the fact that your stepcounter will no longer count steps, and wall layouts are unpredictable. In order to guide people through these voids, you make use of your bottom screen Map overlays (berrymap and roaming map) and walls that block your passage. I can't really make a true guide on how to travel using these Maps, but I can give some recommendations.

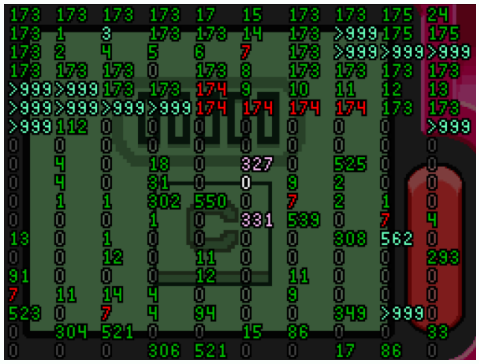
When at Mapdata, by entering Map 2 you can write the value 328 as Map ID.
The location is shown here in the screenshot.



You can also use Map 65, which writes multiple values equal to Battletower Map ID's. These are slightly faster when using a combination of Wrongwarps and battletowers.
Chain in that case is;

Map 173 (static value)
Map 8
Map 82
Map 65
-
3x 327
1 x 328
x 331





Once you're save reset in those, find the coordinate of the spot you wanna be in. I recommend using an open space to give people some leeway here. Once you've guided yourself on the line of one of the coordinates using walls, get to the other using a few steps they have to count themselves, and the poketch Maps.

Next to that, all is up to you. I'm hoping that if you've come this far you can do this! If you need help, I'm always available in my discord mentioned in the Table of Contents.

7. Fake Sinnoh

7.1 What are they

Fake Sinnohs is a very old concept that has been known since the first routes for Darkrai and Shaymin, it's very basic but there are some interesting aspects to it. A Fake Sinnoh, or Fake version of any Map is an occurance where you can Visually see the Matrixdata, despite being nowhere near the Matrixdata. When Saving inside these locations, the game will put you at the same spot in the real Sinnoh, as long as the Map you saved in is an Outdoor Map. But actually, the fact that you see the Fake Sinnoh and that save resetting in them puts you in the Real Sinnoh, is not directly linked to each other.

7.2 Why do they exist

Fake Sinnoh's are actually very easily explainable. All they are is the visual part of the Matrix, which is shown by the Camera. Therefore, we'll have to take a look at how the camera functions.
The camera operates under a 32 bit integer limit. The camera is also so precise that it can focus onto $1/65536^{th}$ of a tile. Since 32 bit can hold 65535^2 amount of values, that means that you can move 65536 tiles in any direction to loop the camera back to 0. At this location the camera will display the visual model of the Map of your current void. Only visual data and heightchanges are assigned to the models, so you can basicly go through all walls as long as a heightchange doesn't stop you from doing so. You can change your heigh by surfing onto water too, if required. Refreshing graphics usually causes the models to be removed, but reloading other areas will show them again.

So why does save resetting in them cause you to end up in the real Sinnoh?
This is tied to the difference between coordinates when moving and how coordinates are saved. When moving, coordinates can go up to the 32 bit integer limit before you loop.
But when you save, your coordinates are stored in 16 bit. Therefore shaving off everything past 65536. If you do Modulo 65536 you'll find the coordinates you'll end up at. Since the camera also loops when your coordinates are past 65536, you'll visually see the exact spot you'll end up at when save resetting.

7.3 (Almost) Obsoleted

Until 2 months ago Fake Sinnohs were completely obsoleted. Due to battletowers, there was almost no reason to use them. Going 65536 steps North goes through highly unpredictable flag-data, which can result in you passing BSOD maps, besides that Battletower routes only took 10-15 minutes and were a better option. But recently I discovered a major breakthrough, being Wrong Warping.
This discovery changed voiding bigtime, allowing us to make almost any route in 5-10 minutes.

By using Wrong Warps you're able to warp straight to Fake Sinnoh's, straight to Mapdata.
There you can either save in a battle tower located inside the Sinnoh Matrix, or manipulate RAM to directly let you save in an outdoor void. Almost all optimised route uses Wrong Warps now.

Read *10. Wrong Warping* for more information.

8. Black Sinnoh

8.1 What are they

Black Sinnoh's are the exact opposite of Fake Sinnoh's. While Fake Sinnoh's are the visual representation of the map while your location in RAM is different, Black Sinnoh's are places in the void where you're in the RAM-section of Sinnoh, but cannot see the models. So you'll find all Map ID's located in the grid of Sinnoh's. These are even simpler to explain than Fake Sinnoh's.

8.2 Why do they exist

If we go back to the basics, we remember that in Outdoor Voids you move 60 bytes when moving up or dawn and 2 bytes when going left or right. This is the cause for Black Sinnoh's. If you've tried routing using Mapdata, you passed one automatically.

Essentially, if you move 60 bytes going up and down and 2 when going left and right that means that going 1 Map down (60 bytes forward) and then moving 30 Maps left (30*2 bytes backwards) causes you to be in the exact same spot in RAM you started of at, but at a different location.
Obviously you can take this to extremes. Moving 1000 Maps down (60 000 bytes forward) and then 30 000 Maps left (60 000 bytes backwards).

You can also apply this to routing to other places obviously, to avoid certain Maps or to be at a specific Y value (this can be useful for certain scripts that check coordinates).

8.3 What are their uses

Despite their simplicity, they're not completely useless. The first use for them is to quickly get to Outdoor Map ID's to assign their location to the Flying option. Then you can get everywhere early. Next to that it also allows for some minor glitches that I won't mention here.

You can also use them to find grass-tiles and watertiles. These will allow you to fight Pokémon here, according to the encountertable of the Map ID you're currently in. This allows you to catch higher level Pokémon early. Graphic reloading can alter Wall-layouts, so find a spot that isn't influenced by them. These are particularly useful when used in combination with *9. Alt-RETIRE routing*.

9. Alt-RETIRE routing

9.1 What is Alt-RETIRE

Alt-RETIRE is a glitch that allows you to call 3e script in runtime. Which means it calls scripts based on your current Map ID. The glitch glitch triggers when you catch 6 Pokémon Migrated from generation 3, which can be done in the void when you enter Pal Park first. Once you enter it, all encountertables are ignored and instead the battles with Migrated Pokémon commences, regardless of Map ID.

9.2 Writing tiles into the void

In the void, tilesections are read from RAM and are therefore manipable to an extent. You can use this to write grasstiles and capture 6 Pokémon in there. Using Mapdata is perfect for that. As I've shown before, this is the wall layout inside Mapdata.

The blue section can contain walls, but also essentially any tile within the game. This includes grass, watertiles, pond tiles, town maps, bins, mud, tall grass, sandtiles, (…)

These tile's vary depending on 2 main important factors. The first one is graphic reloads, by graphic reloading in Maps you are manipulating the tiles that will show up in the top row of the Blue section. Let's say Map 2 (underground) is used. This will write some tiles, one being a grasstile. Sadly we can't open menu's in Map 2, and we can't use Key Items while Pal Park mode is active. Otherwise you can use a townmap or something that reloads graphics, if black sinnohs can't give you the sufficient tiles. The way you influence the tiles is based on your Gfx Set. This is a tileset for your given Map which changes when graphic reloading in a Map. Therefore two maps with the same Gfx Set will give the same tile-changes. I usually reload in Spear Pillar first and then inside any Pokémon Market when I route with Alt-RETIRE. I'm certain there's a faster option tho. (Map 220, Map 4).

The second important factor is movement over loadlines. When you move left, right and down you usually won't influence the tiles. Moving up however shifts the tilesection, you can simply move up twice to revert the shift though. This is the usual course of action; You travel to Mapdata, save reset in Map 392. This will reset all changes to tiles to a constant. Then you travel to Map 393 (Indoor Pal Park) and then Map 251 (Outdoor Pal Park). After that you go to the Black Sinnoh above and find the nearest grasstiles that aren't affected by graphic reloads.

You fight 5 Pokémon there, go to Mapdata, graphic reload with a grasstile writing pattern and then travel to the correct Map ID. Keeping in mind the loadlines. If your Map is located somewhere and the grastile won't show up, I recommend looping the data by going 30 Maps left and then trying again.

Template route;
---
1 S
17 W
14 N
574 W
Save reset

2097 S
19 W
288 S
51 E (above mapdata)
352 S (go through map 2, write map 392)
33 W (go through map 392, write map 393)

Save Reset

1 E
77 S
128 E
19 S (go through map 393, write map 251)
63 E (64 if you have been already)
177 N (enter map 251 / pal park map)
32 N
---
211 W
618 N
19 E
1253 N
14 E (go to grass in black sinnoh)
1 W/ 1 E until captured 5 of the 6 pokémon
---
14 W
1345 S
19 W
---
288 S
64 E
160 S
147 E
18 N
129 W (spear pillar)
graphic reload

After this you reload in any Map with the same ID as PokéMarket.
Do realise that graphic reloading elsewhere will very likely alter your patterns and break the entire route. The grasstile will be located in the top row of any Map, it shouldn't be too hard to find.
Another interesting idea I've been toying with is opening up the entire top row using refreshes, therefore allowing you to go through there avoiding the wall section. I am quite sure I accomplished this once but forgot the pattern unfortunatly, I might look into it later.

10. Wrong Warping

10.1 How Wrong Warping changed voiding as we know it

Welcome to the end of the document, the last and final chapter. This might be my proudest void-related discovery since Alt-RETIRE. By doing a small setup, you're able to warp straight to the last spot you used Explorer Kit. Since Warps teleport you to coordinates stored in 16 bit, you can underflow coordinates, pop up the Explorer Kit textbox and teleport to Fake Sinnoh. You can warp to three different ones. Fake Sinnoh with (FFFF;0000), one with (FFFF;FFFF) and one at position (0000;FFFF). The last two are rather useless since they warp you to a portion of RAM that is filled with 00's. The first one however teleports you straight to Mapdata!

Going vertically Mapdata might be located at 2000-ish steps south, but that means going horizontally takes 60 000 steps, more precise, exactly at where coordinates loop.

This obviously has many applications, but let's start with explaining how to trigger this Wrong Warp in the first place. Earlier in the document I explained how to route to indoor Maps, using Map 7 (a Pokémon Center top floor) as an example. This wasn't just chosen at random, but because this is also the setup route for the Wrong Warp.

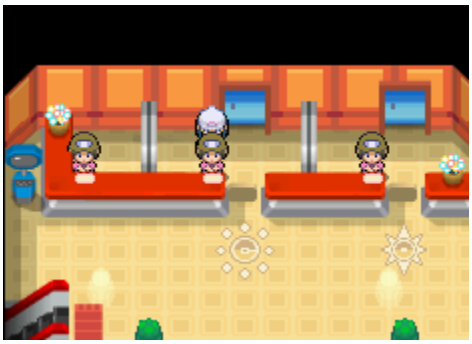
When you usually warp to the Union Room, the game saves your coordinates in 16 bit.
It also sets some flag before saving that tells the game that you're using the Union Room.
This flag causes the game to run a Mapscript when entering any Pokémon Center top floor after that. The Mapscript triggers or when entering the Map. This is important for later.

10.2 Coordinate Warp

There are two things that can be exploited though, with the same cause.
Usually you warp at Y = 0000 0002, but obviously this will be stored as 0002 for warping.
You can make the warp happen higher though, if you talk to the Union Room NPC from above.
This causes the Y value to underflow to Y = FFFF FFFF. Obviously, this will be stored as 0000 FFFF. When you leave the Union Room the game will warp you to the Pokémon Center Top Floor Map 7, but in a Fake Version.

10.3 Wrong Warp

The second exploit is that when talking to the NPC from above, the game doesn't overwrite some bytes it should when talking to her ordinarily. These bytes are used for the coordinates and void to spawn in when you reset the game in the Union Room. So the game fails to write these, where does it warp you to when you reset the game then? These bytes are actually also used for elevator warps and more importantly the Explorer Kit!



To recap, all we have to do to trigger these effects is talk to this NPC from above. Simply let her save the game, and reset the game to warp to your Explorer Kit. By underflowing our X value of the position we use the Explorer Kit first, we can warp us to a Fake Sinnoh inside Mapdata.

You can even combine the two effects of the warps. If you first warp using the Coordinate warp, then leave the void using an elevator, then save with specific assigned data, like a partner for example, you can then reset to warp to your Explorer Kit location but with the partner still active! You can essentially desync your location and any other data this way.

10.4 Routing using Wrong Warps.

We've already established that we can warp to a Fake Sinnoh in Mapdata, this is the Setup I wrote for it ; It puts you just above Mapdata.

1 S
17 W
14 N
415 W

save reset

380 E

Explorer Kit

480 N

260 E

save reset214 W
479 S

graphic reload

Fastbike/running south until wall (you need to go fast to bypass a collisioncheck)

graphic reload
2 S
talk to middle NPC from above

From here, you can very simply route to whatever you want. There are some small limitations tho! First of all, entering any topfloor of any Pokémon center's Map inside the void will crash the game immediately due to the mapscript being ran, which tries to play an animation for an nonexisting door. I provided a separate modified document that marks those Maps too.

But the mapscript can also be used as an 'instant Wrong Warp'!
If you enter the topfloor of any Pokémon Center as intended, the Mapscript will play and move you down a couple of tiles into the void. Then you can do the following steps to get above the NPC again.

4 W
15 N
6 E
graphic reload
Fastbike/running south, until wall
graphic reload
2 S
talk to middle NPC from above
After resetting the game you can route directly to Indoor Maps, save in Outdoor Maps at specific coordinates or get to battletowers quickly. As I mentioned in the Battletower section.

If you run into barriers completely preventing movement here, there are two things you can do.
1) refresh graphics, this might remove the barrier entirely, if they're modelbased.
2) Loop data by going 30 Maps west, where there won't be any barriers

10.5 Map 35 CollisionCorruption routing

Time to abuse Wrong Warps some more. Map 35 as mentioned earlier loads BSOD models which simultaneously give you a semi-walk through walls effect. Wrong Warping can let you warp to Explorer Kit even if you don't actually use it. All you have to do is pop up the message box asking you if you'd like to use Explorer Kit, at this poit your locaton already overwrites the bytes. This was found by my discord Member and loyal glitcher Fleder Kiari. I'd also like to thank MAP, since without him I Wrong Warping might've been completely missed. I was just making him test the Union Room warp, and when he reset he warped to the last place he used Explorer Kit, therefore I could easily figure out the cause and potential uses for Wrong Warping.

You can remove the BSOD effects of Map 35 by losing a battle against Cynthia, so how would you be able to completely abuse this? Well, First you would set up a simple Wrong Warp so you can later use the mapscript based Wrong Warp, just to simplify things.

After that you enter Map 35 using the following route;

1 S
17 W
14 N
607 West

Explorer Kit on Y

save reset

158 W

2097 S

2 W (Enter Map 35)
2 E
978 N
797 E

This will put you under the coordinates of the Sinnoh Matrix.
From here you route towards a specific spot, press Y to activate the Explorer Kit Message and then back out of the message. After that you route to Mapdata, not worrying about any BSOD maps, and enter Map 2. This writes cynthia's Map ID. You enter it, and the battle will automaticly commence because its mapscript based. You lose the battle intentionally, ending up at the Pokémon Center.

Now you enter the top floor and do the usual fast Wrong Warp steps.
Reset the game and voila, you'll teleport to the last spot you used Explorer Kit.

I'm fairly certain I explained everything of importance that is currently being used while routing. There might be some quirks that I missed or future finds could, similar to Wrong Warping and Battle Towers, obsolete some features. Knowing me I probably made a ton of grammatical errors and typo's that I completely missed, but hey, that's me.

I'd like to thank the following people:

Ganix: Where do I start. He has an awesome personality, explained me the basics of voiding and has been supportive of my discoveries. He has helped me many times, and is the person who got me into glitching in the first place by inviting me into the Hall Of Origin. He was and still is a huge inspiration to me.

Aera: Another great person, back in the day he explained to me how the void worked, along with Ganix. I remember when I kept on bothering him to write routes, until he eventually thought me how to use it. Without him I probaply would never have gotten this far.

MKdasher: Although he wasn't active anymore once I joined Hall Of Origin, he has provided me with his void.lua. He even allowed me to share the document, which is the only reason I was even able to make this guide as I could. Without void.lua, which is very user friendly, voiding would be an absolute pain.

MAP: He has been on my side for the past year and helped me with the discovery of Wrong Warping, even if it may be accidentally, many great things came from accidents. Don't take that out of context.

Jagger: He's a great dude who has been updating the German Pokémon sites with my glitches for a while, and is really chill.

Fleder Kiari: A fairly new member to my discord, she's very dedicated and got into voiding immediately. She learned how to void only using this document, so I'm certain you can too! As I mentioned, she discovered that when wrongwarping it isn't required to actually enter the Underground, only to pop up the message box, therefore simplifying the old setup and giving access to Map 35 routing.

BluuBerry: A very chill dude who has been making thumbnails for my youtube channels, he also made my profile picture and background theme. He does an amazing job at his work and truly deserves more recognition, without him my videos wouldn't reach people as they do currently.

Chickasaurus GL, ZZAZZglitch, Crystal_: Although I may not know these people like I do the others, they still inspired me to improve my video quality. They were certainly all influencing my current video style. I recommend checking them out on youtube if you're interested in glitches mainly focusing on gens 1-3.

Sherkel, ISSOtm; Although I don't know these people as well either, both have been supportive in their own way. Sherkel nominated me for Distinguished member on GCL and is a very kind person overall. ISSO may not always be on the same mindset, but even so he influenced me to change my research styles into a more technically oriented manor. That being said, I always try to keep my explanations simple enough for everyone.